Phase-TA: Periodicity Detection and Characterization for HPC Applications

Mathieu Stoffel, François Broquedis, Frédéric Desprez, Abdelhafid Mazouz

Phase-TA: Periodicity Detection and Characterization for HPC Applications

Mathieu Stoffel
Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG
Atos
Grenoble, France
mathieu.stoffel@inria.fr — mathieu.stoffel@atos.net

François Broquedis
Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG
Grenoble, France
francois.broquedis@grenoble-inp.fr

Frédéric Desprez
Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG
Grenoble, France
frederic.desprez@inria.fr

Abdelhafid Mazouz
Atos
Paris, France
abdelhafid.mazouz@atos.net

***Abstract*—The world of High-Performance Computing (HPC) currently stands on the edge of the *ExaScale*. The supercomputers are growing ever more powerful, requiring power-efficient components and ever smarter tool-suites to operate them. One of the key features of those frameworks will be their ability to monitor and predict the behavior of executed applications to optimize resources utilization, and abide by the operating constraints, notably on power consumption.**

In this context, this article presents `Phase-TA`, an offline tool which detects and characterizes the inherent periodicities of iterative HPC applications, with no prior knowledge of the latter. To do so, it analyzes the evolution of several performance counters at the scale of the compute node, and infers patterns representing the identified periodicities. As a result, `Phase-TA` offers a non-intrusive mean to gain insights on the processor use associated with an application, and paves the way to predicting its behavior. `Phase-TA` was tested on a panel of 3 applications and benchmarks from the supercomputing field: `HPCG`, `NEMO`, and `OpenFoam`. For all of them, periodicities, accountable for on average 78% of their execution time, were detected and represented by accurate patterns. Furthermore, it was demonstrated that there is no need to analyze the whole profile of an application to precisely characterize its periodic behaviors. Indeed, an extract of the aforementioned profile is enough for `Phase-TA` to infer representative patterns on-the-fly, opening the way to energy-efficiency optimization through Dynamic Voltage-Frequency Scaling (DVFS).

***Index Terms*—High-Performance Computing, Iterative Applications, Characterization, Periodicity, Software Monitoring and Measurement, Representative Patterns**

I. Introduction

Understanding the behavior of applications running on supercomputers is a key issue that should be handled to further optimize their executions. The profile of an application can be obtained through code analysis and evaluation, but the actual execution of the target application under real conditions is mandatory to get accurate information. Though an application profile can be obtained through static code analysis techniques, actual execution of the target application under real conditions is necessary to get valuable information. And this profile provides the programmer, and/or the parallel execution runtime with important insights. For instance, the fact that an application exhibits different periodicities is extremely useful for further optimizations like the ones related to energy management. In this context, energy minimization can be achieved, for instance by enforcing Dynamic Voltage Frequency Scaling (DVFS) or by controlling the number of active OpenMP threads for each periodic pattern. Such periodic behaviors are illustrated by Fig. 1.

This paper presents `Phase-TA`, a tool able to detect the periodicities associated with the execution of an application, and to build patterns representing them, while being completely agnostic of the aforementioned application, and of its execution environment. The usefulness of this approach is demonstrated on three HPC applications stemming from different application fields, and with different characteristics. The contribution of this paper is twofold: (1) it outlines the methodology implemented by `Phase-TA` to detect periodicities, and infer proxy representations; (2) it presents experimental results, notably showing that a panel of HPC applications exhibit periodicities dominating most of their execution time.

The remainder of the paper is organized as follows. Section II presents the overall methodology of `Phase-TA`, together with some preliminary definitions. The next section (III) provides information about the behavior of `Phase-TA`: how periodic instances are detected, extracted, and clustered, and how representative patterns are inferred. Section IV presents the use of `Phase-TA` on various applications with different behaviors, and section V gives both a theoretical study of its complexity, and an empirical evaluation of its performance. Then, the section VI presents some use cases for `Phase-TA`, along with what motivated its development, and the paths to be explored in future work. Finally, after a section (VII)

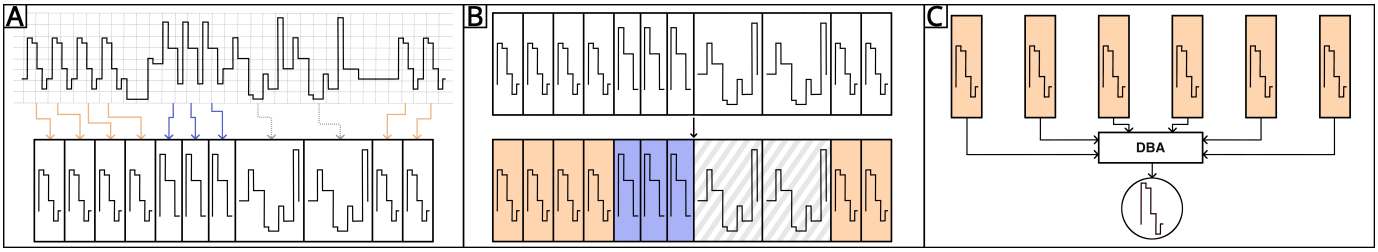

Fig. 1: Schematic description of the three main steps of the methodology underlying Phase-TA: (A) detection of periodic instances (B) clustering of periodic instances (C) inference of representative patterns from clustered periodic instances.

presenting existing related works from the literature, the last section (VIII) draws some essential conclusions.

II. Methodology of Phase-TA

Firstly, this section defines some terms and concepts which are essential to understand the remainder of this article. Secondly, it offers a coarse-grain overview of how the application profiles are obtained, and, then, of the methodology implemented by Phase-TA.

A. Preliminary Definitions

To begin with, an **application profile** represents the temporal evolution, induced by the execution of an application, of a measurable and/or estimable quantity, *at the scale of a compute node*. For instance, let's suppose that the number of bytes sent by a node through its network interface is monitored during the execution of an application. Then, the number of bytes sent through the network as a function of the time spent since the execution of the application has started is an application profile. In other words, an application profile is a time series, which points are also called **samples**.

An application profile may feature **periodicities**, that is to say minimal patterns which regularly occurs. Consecutive occurrences, at least two, of a periodicity constitute a **periodic region**. Within a periodic region, each occurrence of the periodicity is called a **periodic instance**. By way of example, let's consider a sequence on the set $\mathcal{S} = \{A, B, C, D\}$, and, for the sake of clarity, let's only consider periodicities of length at least 2. As it can be seen below, the aforementioned sequence features two periodicities, namely AAB and CBA, which respectively induce 2 and 1 periodic regions, delimited by square brackets. Those periodic regions contain respectively 4 and 3 periodic instances, identified by curly braces, respectively below and above the sequence:

$$[\underbrace{AAB}\,\underbrace{AAB}]BD[\overbrace{CBA}\,\overbrace{CBA}\,\overbrace{CBA}]\,\text{AAB}\,DC[\underbrace{AAB}\,\underbrace{AAB}]$$

It should be noticed that the orange sub-sequence of characters is not a periodic instance, since it does not belong to a periodic region. This aspect of the definition of a periodic instance might seem counter-intuitive. It stems from the way periodic instances are detected by Phase-Ta, which is detailed in the section III-A.

For discrete-valued application profiles, periodic instances are exact copies of the periodicities from which they stem. However, for real-valued application profiles, in some cases, two periodic instances associated with the same periodicity may slightly differ (for example, consider the slight local heterogenous noise variations affecting the measurement time series of a sensor). As a result, it is impossible to exactly identify the periodicity inducing the observed periodic instances. One very standard approach consists in averaging the aforementioned periodic instances to build a proxy representation of the periodicity. In the remainder of this article, such a proxy is called a **representative pattern**.

Finally, let's define the "flavor" of the **Within-Group Sum of Squares** (WGSS for short) function being used by Phase-TA to analyze real-valued application profiles. Let $\mathcal{I} = \{i_n\}_{n\in\mathbb{N}}$ be a set of periodic instances, and p be a representative pattern associated with $\mathcal{I}$. Then, the WGSS associated with p, relatively to $\mathcal{I}$ is defined by:

$$wgss(p, \mathcal{I}) = \sum_{i\in\mathcal{I}} DTW_2(p, i) \tag{1}$$

Note that in (1), DTW (and DTW_2, which is its squared version) refers to *Dynamic Time Warping* [1], a similarity measure particularly well-suited for real-valued time series featuring temporal variations. Some more details about DTW:

- DTW is computed between two time series. It creates pairs of points/samples: each sample of one of the two input profiles is matched with at least one sample of the other profile.
- Two paired points are said to be aligned, and the DTW measurement is equal to the sum of the pointwise distances between each pair of aligned samples.

To finish, DTW_2 is defined in the same way as DTW, the only difference being the fact that the **pointwise euclidean distance** between paired points is **squared**. In other words, $DTW_2 \neq DTW^2$, since the computation of DTW involves a sum, and this is not the sum which is squared, but each of the summed terms individually.

B. Obtaining applications' profiles and overview of Phase-TA Methodology

To begin with, let's focus on the application profiles analyzed by Phase-TA. Those profiles are generated by a lightweight daemon, later called *profiler*. This profiler is part

of BDPO [2], an HPC tool aiming at improving the energy-efficiency associated with the execution of HPC applications by leveraging fine-grain monitoring. It uses libpfm [3] to periodically sample several performance counters, which are sampled and aggregated every 5 ms, at the scale of **the whole node, for all processes**. As a result, one instance of the profiler is executed on each node involved in the execution of the considered application. This instance produces the application profile associated with the node on which it is executed. Also, it was verified that the impact on the application profile of the execution in the background of the profiler was negligible [2].
Now, let's focus on Phase-TA with Fig. 1, which presents the three main steps of the analysis methodology it implements. The first step consists in a pass over the whole application profile to detect and extract the periodic instances. Then, since those periodic instances might be related to different periodicities, it is necessary to cluster them. Finally, the periodicities are averaged, per cluster, so as to build a representative pattern per periodicity. The section III gives the specifics of those three outlined steps.

III. Architecture of Phase-TA

This section details the implementation of the three main parts of Phase-TA methodology, as presented by section II-B.

A. Detecting and Extracting the Periodic Instances

An application profile can be represented as a 1D horizontal vector, which elements corresponds to the samples belonging to the aforesaid profile. Let's suppose that, chronologically, the first sample of the profile, of ID 0, is at the left end of the considered vector. Thus, sample IDs increase toward the right end of the vector.
The profile is analyzed piecewise, using a sliding window (SW) divided in two parts of same length L (i.e. L-sample long), respectively named left (LSW) and right (RSW) sliding windows. SW slides towards the right in the course of the analysis, taking $T \in \mathbb{N}^*$ positions. For all $t \in [\![0; T[\![$, SW^t denotes the t-th position of SW, and LSW^t and RSW^t respectively denote its left and right parts.
From now on, let's consider a fixed $t \in [\![0; T[\![$.
To extract the potential periodic instances from the considered SW^t, an algorithm derived from the Dynamic Periodicity Detector, presented by Freitag et al. in [4], was implemented.
As illustrated by Fig. 2, for all $k \in \mathcal{A} = [\![1; L-1]\!]$, $\text{SW}^t_k = \text{SW}^t >> k$ is computed, where $>>$ refers to the right-shift operator. Note that SW^t is **not** slided, its content is shifted. In other words, SW^t can be considered as a 1D horizontal vector which content is fixed, and SW^t_k as a copy of SW^t which content is modified by applying a right-shift. k is, hereafter, also referred to as the *sliding factor*. Then, the euclidean distance between RSW^t and RSW^t_k, termed as $d_2(\text{RSW}^t, \text{RSW}^t_k)$, is evaluated for all $k \in \mathcal{A}$. The underlying rationale is that if $d_2(\text{RSW}^t, \text{RSW}^t_k)$ is close to 0, then RSW^t can be considered as k-periodic [4].
Thus, by looking for the global minimum of $d_2(\text{RSW}^t, \text{RSW}^t_k) = f(k)$, it is possible to determine whether or not RSW^t exhibits a periodicity. Or, more precisely, as shown by Fig. 2, a family of periodicities. Indeed, if $d_2(\text{RSW}^t, \text{RSW}^t_k)$ is minimal and close to zero for $k_* \in \mathcal{A}$, then RSW^t is k_*-periodic. But it might also be p-periodic, for $p \in \mathcal{A}$ such that p is a multiple of k_*, or k_* is a multiple of p. By considering the other values of k for which $d_2(\text{RSW}^t, \text{RSW}^t_k)$ is close to 0, it is possible to determine the *base periodicity*, named k_{bp}, that is to say the lowest periodicity of the detected family of periodicities.
Finally, $\lfloor L/k_{bp} \rfloor$ periodic instances are extracted from RSW^t, and SW^{t+1} is obtained by sliding SW^t to the right by $k_{bp} \cdot \lfloor L/k_{bp} \rfloor$ samples.

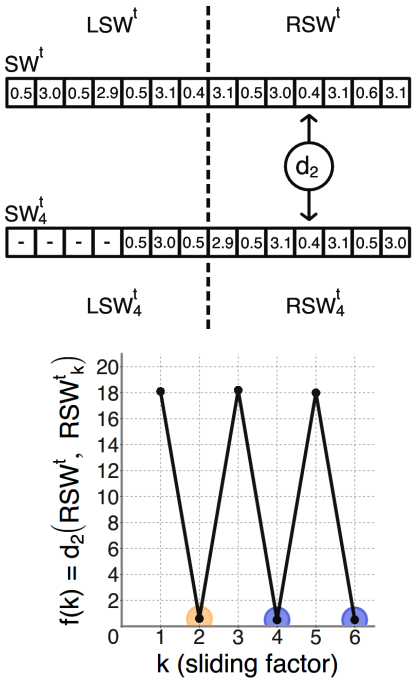

Fig. 2: Example conceptually illustrating the periodicity detection algorithm implemented by Phase-TA. At the top, an example of sliding window, and its version right-shifted by 4. At the bottom, the graph related to $f(k) = d_2(\text{RSW}^t, \text{RSW}^t_k)$, featuring the detected family of periodicities (circled), and the associated base periodicity (in orange).

As it can be intuited, the length of SW has an impact on the periodicity detection. For instance, with a length of $2L$, it is possible to detect periodicities of length at most L. Moreover, the optimal value of L is, a priori, dependent on the application, and finding a proper value for L seems not to be a trivial task.
That is why Phase-TA auto-tunes the value of L. A rather wide initial search interval for L is roamed in a dichotomic-alike manner, detecting and extracting the periodic instances for each visited value of L. The objective function to maximize is the percentage of the whole profile for which the extracted periodic instances are accountable. In the context of this

article, the search interval is $[\![0; 10000]\!]$ samples.

B. Clustering the Periodic Instances

First, the detected periodic instances are grouped according to their lengths, so as to speed up their clustering (which complexity is a function of the number of clustered elements). This pre-processing step relies on the fact that in order for two periodic instances to be related to the same periodicity, they must have similar lengths, both in the neighborhood of the length of the aforesaid periodicity. In order to filter out the outlier and marginal periodic instances, the groups which does not represent more than α percents of the whole application profile are dropped (in the context of this article, $\alpha = 5\%$).
Then, since an application profile might exhibit several distinct periodicities which lengths are similar, or even equal, a clustering algorithm is applied to each group. As a result, the periodic instances belonging to a cluster should be related to the same periodicity.
In the context of `Phase-TA`, several clustering algorithms were evaluated:

- OPTICS [5], using DTW as similarity measurement between two periodic instances. Several heuristics to set its parameters (ϵ and $MinPts$) were tried;
- K-means after applying feature extraction based on statistics to the periodic instances;
- Aggregative hierarchical clustering, using DTW as similarity measurement between two periodic instances. The single and complete linkage criteria were evaluated.

In order to assess the efficiency of those clustering algorithms, the periodic instances detected by `Phase-TA` for profiles associated with the three applications (namely HPCG, NEMO, and OpenFOAM) of the experimental panel presented by section IV-B were clustered and labeled by hand (one profile per application). Thus, it was possible to compute the F_1-score associated with each clustering algorithm.
On average, aggregative hierarchical clustering with single linkage criterion achieved the top F_1-score, and was therefore chosen as the clustering algorithm implemented by `Phase-TA`.

C. Building the Representative Patterns

To build the representative pattern associated with a periodicity, `Phase-TA` averages the periodic instances belonging to the related cluster. To this end, it uses Dynamic time warping Barycentric Averaging (DBA), an algorithm detailed by Petitjean et al. in [6]. To sum it up, it consists in a refinement process based on the DTW similarity measurement. DBA notably leverages the fact that DTW creates pairs of aligned points (see the brief definition of DTW in section II-A). Starting from an arbitrary initial average, each iteration of the algorithm modifies the pattern so that its WGSS relatively to the periodic instances being averaged should decrease. In fact, it is **guaranteed**, and demonstrated in [6], [7], that the WGSS can **only decrease** when a refinement step is performed. In the same way, it is proven that the time series minimizing the WGSS relatively to the periodic instance is unique, and that the refined pattern asymptotically approximates the former when the number of iteration of DBA increases.
Let's detail the sub-step performed by **each iteration** of the DBA algorithm:

1) Computing the DTW measurements between the average and each periodic instance;
2) For each point $\mathcal{P}$ of the average, building the set $\mathcal{S}$ of all the points of the periodic instances with which $\mathcal{P}$ was paired, according to the DTW;
3) Computing the barycenter of $\mathcal{S}$, which is the refined (i.e. new) value of $\mathcal{P}$ at the end of the considered refinement step.

Just as for any refinement process, the initialization and termination criteria are of uttermost importance, notably impacting the convergence speed.
Concerning the termination criterion, the refinement process implemented by `Phase-TA` terminates if either one of the two below conditions is met:

- N_{iter} refinement iterations were performed (in the context of this article, $N_{iter} = 31$);
- n_{cons} consecutive refinement iterations induced a relative decrease of the WGSS associated with the representative pattern lesser than t_{term} (in the context of this article, $n_{cons} = 5$ and $t_{term} = 2.5\%$).

The parameters related to the termination criterion were set to values arbitrarily, based on empirical observations.
Concerning the initialization criterion, in [6], Petitjean et al. offer some guideline concerning the initialization of the pattern: DBA tends to converge faster when initialized with one of the averaged time series. Since the DTW measurements between each pair of periodic instances belonging to the considered cluster were computed at the clustering step (see section III-B), `Phase-TA` initializes the DBA algorithm with the periodic instance minimizing the within-group sum of DTW measurements. To make sure computing the aforementioned sum was worth, this initialization scheme was empirically compared to selecting randomly one of the periodic instance. For the panel of applications considered in this article (see section IV-B), it was observed that, on average, the initialization scheme of `Phase-TA`:

- Lead to faster convergence of the refinement process (i.e. the termination criteria of the DBA algorithm implemented by `Phase-TA` was met after a lesser number of iterations);
- Resulted in representative patterns which associated WGSS were lesser that their counterparts obtained with random initializations.

IV. Experimental Validation

To begin with, the experimental setup and methodology will be detailed. Then, several experiments will be presented, and their results discussed.

A. Experimental Setup

All the experiments presented in this article were performed on 16 compute nodes of the `dahu` cluster of the Grenoble

facility of Grid'5000 [8], an experimental testbed supporting computer science research in France. The main characteristics of one of the compute nodes of `dahu` are listed in Table I.

TABLE I: Characteristics of one compute node of the `dahu` cluster

Processors	Intel Xeon Gold 6130
Number of sockets	2
Number of cores	32 (2×16)
Memory	DDR4@2.67 GHz
Memory size	192 $GB(6 \times 32\ GB)$
Network	Intel® Omni-Path 100 Gb/s
OS	RedHat RHEL 7.4

Hereafter in this article, the **featured application profiles represent the number of Instructions retired Per reference Cycle (IPC)** (ratio `INSTRUCTION_RETIRED` / `RDTSC`), and were generated by the profiler introduced by the section II-B However, other profiles were studied (e.g. the evolution of the number of `L3` cache misses per reference cycle), and yielded similar results.
Finally, all the experiments presented in this paper were **reproduced at least 3 times**, using the default configuration of `Phase-TA`, and produced similar profiles for each execution, on all of the involved nodes.

B. Displaying the Periodicities of HPC Applications

This set of experiments aims at demonstrating that iterative HPC applications exhibit periodicities, accountable for a significant part of their execution time. To do so, 3 well-known applications and benchmarks from the supercomputing field were executed: the GYRE component of NEMO [9], an ocean modeling framework; the HPCG [10] performance benchmark built upon the conjugate gradient computing recipe; and the icoFOAM solver, of the OpenFOAM CFD toolbox [11], applied to the cavity test case. As it can be noticed, the selected panel contains applications from the most common fields of scientific computing, exhibiting various behaviors from compute to memory boundness. Those applications were executed on sixteen compute nodes, for a total of 512 cores. On the one hand, NEMO and OpenFOAM were run with 512 MPI ranks, each one pinned to a specific core. On the other hand, HPCG was run with 1 MPI rank per socket, and 16 OpenMP threads per MPI rank, for a total of 512 OpenMP threads. Similarly, both the MPI ranks and the OpenMP threads were pinned to specific cores. Finally, let's precise that the associated configurations were tuned, and the workloads seized so that the performance achieved on the partition of 16 compute nodes should be "locally optimal" (i.e. optimal in a reduced search space, narrowed based on empirical observations).
The produced per-node IPC profiles were analyzed by `Phase-TA`, and the results are summarized in Table II, where: (1) `T_exe` is the execution time of the application in seconds; (2) the `#rp(#i)[#pa]` column contains: (a) the number of representative patterns `Phase-TA` inferred, (b) between brackets, the total number of periodic instances associated with those patterns, and (c), between square brackets, the average number of consecutive related periodic instances within a periodic area; and, finally, (3) `%period` is the proportion of the whole profile covered by the periodic instances from which the representative patterns were inferred. By way of example, the representative pattern inferred by `Phase-TA` for a profile of an execution of NEMO is displayed by Fig. 3.

TABLE II: Analysis of the IPC profiles of several HPC applications

Applications	`T_exe`	`#rp(#i)[#pa]`	`%period`
HPCG	1129.52	3(521)[14]	89.20%
NEMO	659.696	1(451)[21]	76.73%
OpenFOAM	659.391	1(169)[4]	67.48%

To begin with, it can be observed that all the applications exhibited periodicities for which `Phase-TA` inferred representative patterns, accountable for on average 77.80% of the whole application profiles, and even up to almost 90% for HPCG.
Let's analyze the results further, starting with NEMO. The whole execution time of NEMO is almost dedicated to computation: a computational kernel is applied to a 101 levels 2D-grid, for a given number of timesteps to be simulated. As a result, the computation phase induced a single periodicity, accountable for more than 75% of the execution time.
Concerning HPCG, it exhibits three patterns: the first two are associated with the initialization of data, accounting for about 31.8% of the profile, and the third one with the computation, accounting for about 57.4% of the profile. Indeed, HPCG programmatically generates its dataset during its initialization step, which coarsely accounts for 40% of the total execution time of HPCG. Consequently, patterns are inferred for the aforementioned data initialization, which is partially iterative.

On the contrary, icoFOAM got its input dataset from files, and the process appears to be aperiodic. On top of that, the execution of icoFOAM contains checkpoints after each bunch of computation steps (to be precise, each run of icoFOAM contained five checkpoints). Those checkpoints seem to be aperiodic I/O phases, and account for approximately 25.0% of the total execution time of the application. Consequently, `Phase-TA` infers two representative patterns. The remaining of the execution time is dedicated to computation, for which `Phase-TA` inferred one representative pattern accountable for more than 67% of the execution time, which corresponds to roughly 89% of the execution time associated with computation.
For the three aforementioned applications, one additional important observation should be made. The portions of the profiles which are not accounted for by the above explanations are most probably periodic instances which `Phase-TA` failed to detect. Indeed, even if the workflow it implements allow for some perturbations, both spatial (i.e. fluctuations of the sampled values) and temporal (i.e. insertions and/or deletions of samples), some periodic instances are too noised to

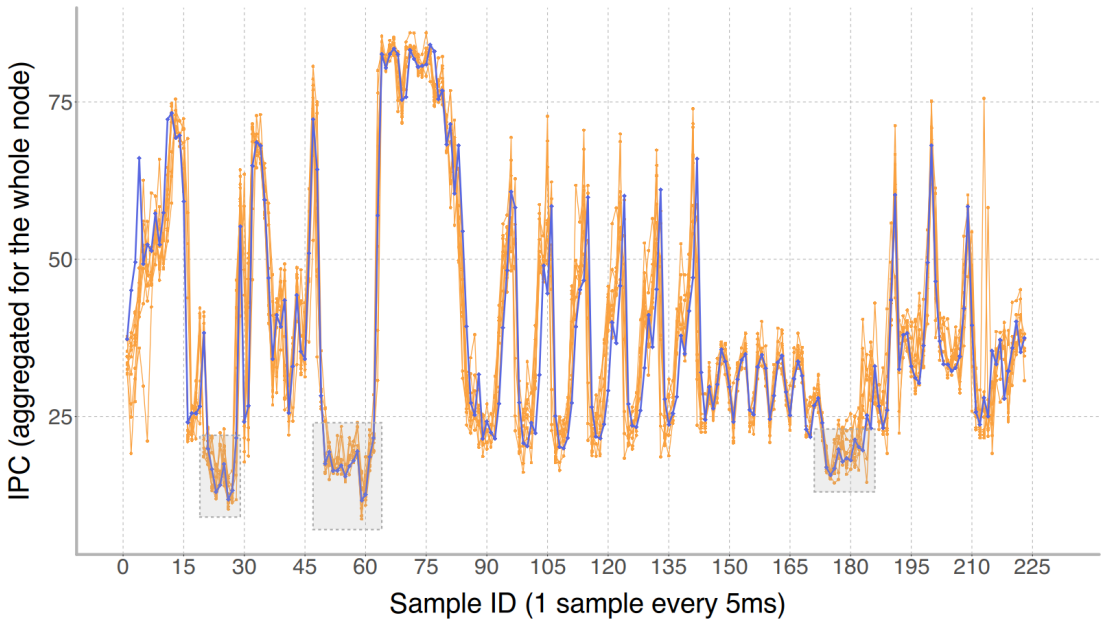

Fig. 3: Representative pattern (in purple) associated with NEMO, overlaid on a random selection of ten of the periodic instances (in orange) it was inferred from. Gray areas indicate regions of interest which will be referred to in section VI.

be detected by `Phase-TA` with its default configuration. Note that, by tuning the configuration of `Phase-TA` per application, it is possible to detect the aforementioned missed periodic instances. However, it requires an expertise the standard HPC user is not likely to have. That is why the default configuration was designed to be as much application-agnostic as possible, and prefers missing a few periodic instances rather than detecting several false positives.

Let's focus on the representative patterns inferred by `Phase-TA`, such as the one displayed by Fig. 3. To begin with, as previously mentioned, a representative pattern is inferred relatively to a particular profile. In the context of this article, a profile is specific to one compute nodes. As a result, for an execution of application on 16 compute nodes, there are 16 different profiles, for which potentially different representative patterns can be inferred by `Phase-TA`.

However, as it is shown by Fig. 4, it appears that, and it is true for all of the three applications, the inferred representative patterns are almost identical for the groups of 16 related profiles. It is confirmed numerically: for each pair of related inferred patterns (e.g. the patterns inferred for the profiles associated with `dahu-1` and `dahu-2`, for one execution of NEMO), $\frac{1}{assoc\_DTW} \cdot DTW \leq 0.01 \cdot mean(patterns)$. It coarsely means that the average distance, according to the DTW, between two related points of two representative patterns is lesser than 1% of the average value of the points belonging to the related representative patterns.

Those experiments showed that the portion of the applications' profiles associated with periodicities depends on the specificities of the considered applications. Nonetheless, for those experiments, at least two thirds, and on average more than 75%, of the execution time of the considered applications were attributable to periodicities, which is **significant**. Finally, those experiments showed that the **inferred representative patterns** are **relevant** relatively to the analyzed profiles, and that they can be used as **proxy** for a large part of the latter.

C. Influence of the Number of Detected Periodic Instances on the Relevance of the Inferred Representative Pattern

`Phase-TA` analyzes the profiles associated with the execution of HPC applications. However, the aforementioned executions generally take at least several hours, often several days, to complete. Keeping in mind that the sampling frequency of the profiler is $200Hz$, it appears that a whole profile can easily contain billions of samples. Thus, the following questions can be raised: would **considering only a sub-profile**, extracted from the whole profile, be a conceivable tradeoff to infer relevant representative patterns, while reducing the workload associated with the analysis? What should be the **minimum duration** of a profile so that the inferred pattern may be **relevant**?

In order to answer those questions, an experimental protocol was designed and implemented for the applications of the experimental panel. To begin with, a *long run* (around one hour long) of each application was executed. To do so, the same workloads as for the experiments presented by the section IV-B was used, but the number of iterations was increased.

Then, the long run profiles were analyzed by `Phase-TA`. The inferred representative patterns are called *reference patterns* in the remainder of this paragraph.

Next, sub-profiles were extracted from the long run profiles, and analyzed:

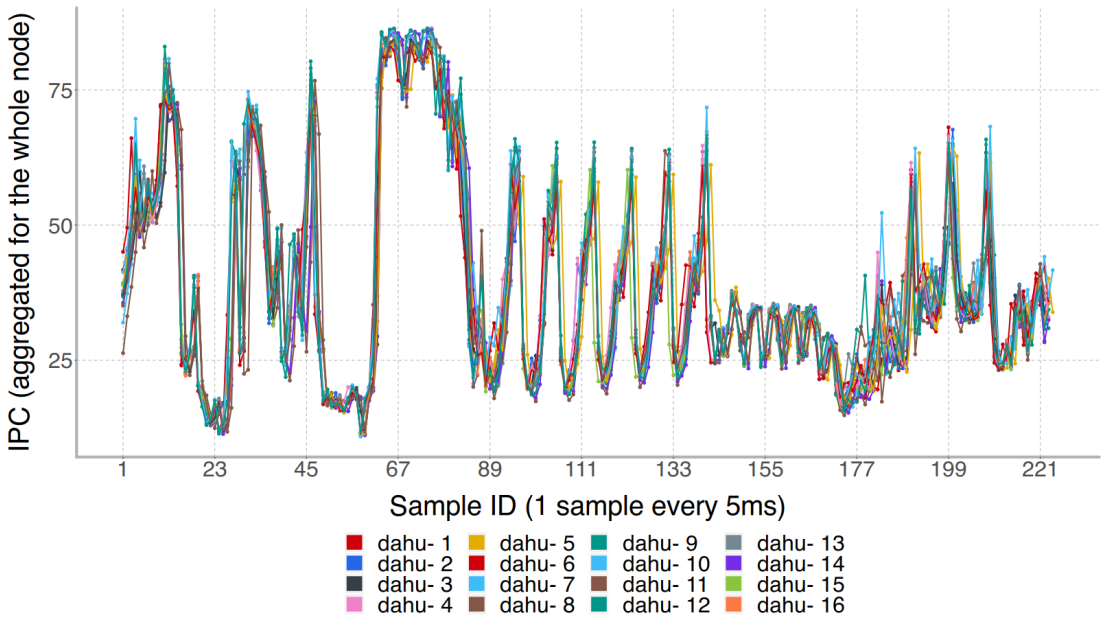

Fig. 4: Representative patterns inferred from the profiles related to the 16 compute nodes, for the same execution of NEMO

- Several durations were considered (in minutes): 1, 3, 5, 10, 15, 30, and 45; Hereafter, it is important to keep in mind that the profiles result from monitoring performance counters with a sampling period of $5ms$;
- For each duration, 11 different sub-profiles were extracted, with different randomly selected offsets, relative to the beginning of the associated whole profile;
- It was enforced, and verified, that the randomly selected offsets were long enough so that the sub-profiles should not overlap with the initialization steps of the concerned applications.

The representative patterns inferred for the sub-profiles are, hereafter, called *sub-profile patterns*.

In the remainder of this paragraph, only the representative patterns with the highest *coverage score* (i.e. the proportion of the whole profile represented by the periodic instances related to the pattern) are considered, for both the sub-profiles and the long runs profiles. It was verified that, per application, the sub-profiles patterns represented the **same** periodicity as the related reference patterns.

Let $\mathcal{P}$ be a long run profile, r be the reference pattern inferred for $\mathcal{P}$, and $\mathcal{I}$ be the set of periodic instances associated with r. Next, for any sub-profile $\mathcal{S}$, let s be the associated sub-profile pattern. Now, let the *sub-profile* **Within-Group Sum of Squares (WGSS)** be defined as $wgss(s, \mathcal{I})$, and the *reference WGSS* be defined as $wgss(r, \mathcal{I})$. In other words, the sub-profile WGSS is associated with the sub-profile pattern, relatively to **all** the periodic instances from which the **related reference pattern** was inferred. Then, by comparing a sub-profile WGSS to the related reference WGSS, the relevance of the associated sub-profile can be assessed. Therefore, it is possible to **evaluate the relevance of sub-profile patterns as a function of the duration of the related sub-profiles**.

Fig. 5 presents sub-profile WGSS as a function of the duration of the associated sub-profile. Each cross represents the WGSS associated with a sub-profile, while diamonds and error bars respectively represent per-duration average and standard deviation associated with sub-profile WGSS. Next to each diamond is indicated the average number of periodic instances for a given duration. Finally, horizontal blue lines represent the reference WGSS for the considered applications.

First, let's notice that even for the shortest sub-profiles, the relative difference between sub-profile WGSS and reference WGSS is **acceptable**: for the 3 applications, the former is on average 37.0% greater than the latter (up to 67.3% for OpenFOAM).

Moreover, it should be observed that the number of detected periodic instances greatly varies, depending on the application: for a 10-minute sub-profile, it ranges from 145 for OpenFOAM, up to 404 for NEMO. Thus, when it comes to the relevance of the inferred sub-profile patterns, it seems that the number of periodic instances detected for the concerned sub-profile is more significant than the latter's duration.

Let's try to define a threshold on the number of periodic instances, above which an inferred representative pattern can be regarded as relevant. From Fig. 5, 100 seems to be a sound candidate. Indeed, on the one hand, when considering the sub-profile durations for which the numbers of detected periodic instances is the closest to 100 (while being greater or equal to 100), the sub-profile WGSS is on average only 12.1% greater than the reference WGSS (up to 24.4% for NEMO). On the other hand, this number of periodic instances is reachable with a rather short profile. Indeed, when considering the three applications of the panel, the average profile duration required to reach 100 periodic instances is 6 minutes, and at most 10 minutes, which is a short duration when compared to HPC

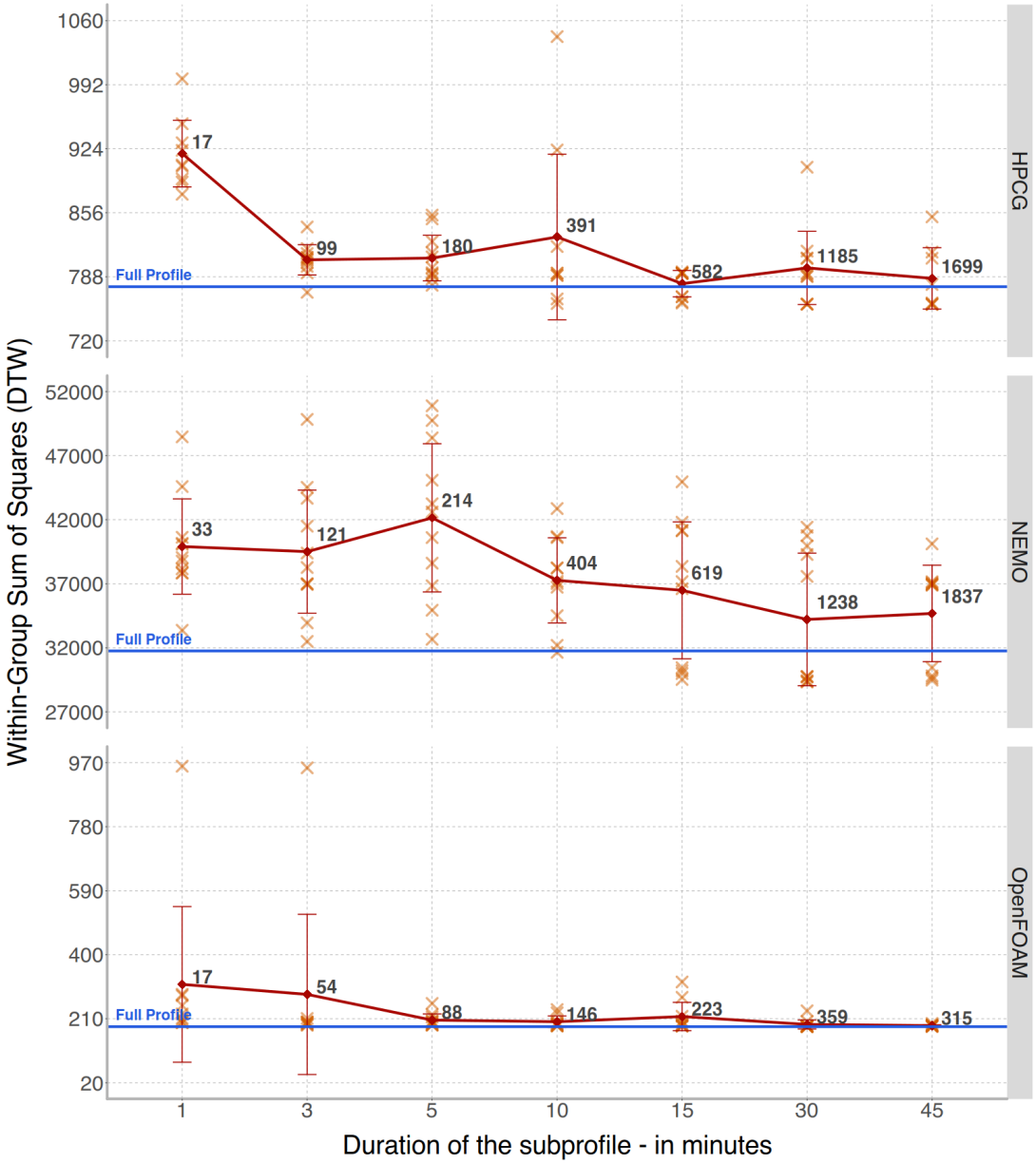

Fig. 5: WGSS associated with sub-profiles as a function of their durations, for HPCG, NEMO, and OpenFOAM.

jobs lasting for several hours, nay days. As a result, it can be retained, as a rule of thumb, that representative patterns inferred from a 10-minute-long sub-profiles are very likely to be relevant for the whole profile.

To be thorough, it should also be observed that: (1) some sub-profiles lead to sub-profile WGSS lesser than the related reference WGSS, and (2) sub-profile WGSS does not necessary decrease and/or tend to reference WGSS when sub-profile duration increases, **once the aforedefined threshold is crossed**. The underlying explanations to those two observations lie with the definition of the DBA algorithm. As said in the section III-C, their is no guarantee concerning the convergence speed associated with the refinement process of the average. As a result, depending on the initialization criterion, and the cluster of periodic instances, the decrease of the WGSS induced by a refinement step may vary erratically. Furthermore, the termination criterion of the refinement process is **sensitive** to the convergence speed. This can lead to different numbers of refinement steps for different sub-profiles, even if they have the same duration. Put together, those elements explains the two aforementioned observations.

To summarize the results of this experiment, two conclusions should be drawn: (1) it is a **conceivable tradeoff** to consider **only an extract of a profile** to infer the associated representative patterns, and (2) as a rule of thumb, representative patterns can be regarded as **relevant** when they are inferred

from **at least 100 periodic instances**, which is usually met for a profile duration of **at least 10 minutes** (for application profiles generated with a $5ms$ sampling period).

V. Complexity Study and Performance Evaluation

This section regroups a theoretical complexity study, and an empirical performance evaluation, of the analysis workflow implemented by `Phase-TA`.

A. Complexity Study

First of all, let's suppose that a profile of length $\mathcal{L}$ (i.e. containing $\mathcal{L}$ samples) is analyzed by `Phase-TA`, which detects N_{pi} periodic instances of average length L_{pi}. On top of that, let's suppose that the configured length of the sliding windows, and maximum number of iterations for DBA are respectively L_{SW} and I_{DBA}. Then, let's note C_{arith} the complexity of a generalized arithmetic operation (e.g. an addition, an absolute value, a multiplication, ...), and C_{clust} the complexity of a generalized clustering operation (typically constituted of a fixed set of generalized arithmetic operations and data structure management operations). Finally, let's note C^x_{DTW} the complexity of computing the DTW between two periodic instances of length x, and $C^x_{N_1}$ the complexity of computing the distance of Manhattan between two periodic instances of length x.

As a result, the complexity of the workflow implemented by `Phase-TA`, noted C_{phase}, is:

$$\begin{aligned} C_{phase} = & \mathcal{O}\left(\left[\left(\mathcal{L}\cdot\frac{1}{L_{SW}}\right)\cdot\left(C_{N_1}^{L_{SW}}\cdot L_{SW}\right)\right]\right) \\ + & \mathcal{O}\left(\left[\frac{N_{pi}\cdot(N_{pi}-1)}{2}\cdot C_{DTW}^{L_{pi}} + N_{pi}^2\cdot C_{clust}\right]\right) \\ + & \mathcal{O}\left(\left[I_{DBA}\cdot\left(N_{pi}\cdot C_{DTW}^{L_{pi}} + L_{pi}\cdot C_{arith}\right)\right]\right) \end{aligned}$$

The first term between square brackets is associated with the complexity of the periodicity detection step (see section III-A). Its left sub-term represents the fact that the profile is analyzed one sliding window at a time, and its right sub-term represents the complexity of computing the Manhattan distance between the sliding window and each of its right-shifted counterparts. The second term between square brackets is associated with the complexity of the clustering step (see section III-B). Its left sub-term is associated with the complexity of computing the DTW-based distance matrix for the periodic instances to be clustered. Its right sub-term represents the complexity of the clustering algorithm itself, and is demonstrated in [12].

The third term between square brackets is associated with the complexity of the DBA step (see section III-C), and the its expression is demonstrated in [6].

Knowing that $C_{N_1}^{L_{SW}} = \mathcal{O}\left(L_{SW}\cdot C_{arith}\right)$ (trivially, from its definition [13]), and that $C_{DTW}^{L_{pi}} = \mathcal{O}\left(L_{pi}^2\cdot C_{arith}\right)$ (demonstrated in [1]), the complexity of the analysis workflow of `Phase-TA` can be rewritten as:

$$\begin{aligned} C_{phase} = \mathcal{O}([\mathcal{L}\cdot L_{SW} + N_{pi}^2\cdot L_{pi}^2 + I_{DBA}\cdot N_{pi}\cdot L_{pi}^2 + \\ I_{DBA}\cdot L_{pi}]\cdot C_{arith} + N_{pi}^2\cdot C_{clust}) \end{aligned} \quad (2)$$

From (2), it appears that C_{phase} depends on two configuration parameters, namely L_{SW} and I_{DBA}, and three linked characteristics of the analyzed profile, namely $\mathcal{L}$, N_{pi}, and L_{pi}. On top of that, the complexity is **globally coarsely quadratic relatively to the length of the profile**, since $\mathcal{L}$ is an upper bound for $N_{pi}\cdot L_{pi}$, and that, in the ideal case, the latter tends toward the former. Since N_{pi} and L_{pi} are linked, since when one increases the other one decreases proportionally, and vice versa, the following performance evaluation should notably make it possible to determine if one those two parameters is the dominating factor regarding the complexity.

B. Performance Evaluation

In order to empirically evaluate the performance of `Phase-TA`, measurements of its execution time were performed. Both the number of detected periodic instances, and their lengths, have an impact on the algorithmic complexity associated with `Phase-TA`. The three applications of the experimental panel exhibit periodic instances with lengths varying from around 220 samples for NEMO, to more than 520 samples for OpenFOAM; and varied numbers of periodic instances, as shown by Table II. Thus, this empirical evaluation should make it possible to assess the impact of those two factors on the performance of `Phase-TA`.

The implemented protocol is described by the points below:

- For each of the selected applications, each of the sub-profiles extracted for the experiments presented by the section IV-C was analyzed 5 times;
- The complete set of measurements was performed on two different nodes of the `dahu` cluster (presented in section IV-A), and yielded almost identical results (the relative difference is lesser than 1%).

Table III presents the results associated with the evaluation of the performance of `Phase-TA`. Each row is associated with one of the selected applications, and each column is associated with a sub-profile duration, in minutes. Each cell contains the average execution time of `Phase-TA` for a given sub-profile duration. To be precise, first, the average execution time of the 5 analysis performed for each sub-profile is computed. Then, those per sub-profile average execution times are averaged per duration, giving the values reported by Table III.

TABLE III: Execution times (in seconds) of `Phase-TA`, relatively to the duration of the analyzed sub-profile (in minutes)

	1	3	5	**10**	15	30	45
HPCG	0.44	1.18	2.67	**9.33**	19.60	76.00	168.84
NEMO	0.45	1.16	2.67	**8.73**	18.38	71.13	162.76
OpenFOAM	0.84	1.40	2.62	**6.95**	13.36	46.98	73.69

As it can be observed, the execution time of `Phase-TA` increases polynomially relatively to the duration of the analyzed profile (more precisely, relatively to the number of detected periodic instances, which tends to be proportional to the duration, as shown in Fig. 5). Additionally, it should be noticed that for shorter sub-profiles, when the number of periodic instances is small, their lengths is the dominating factor when

it comes to the performance of Phase-TA (OpenFOAM exhibits less but longer periodic instances when compared to NEMO and HPCG). On the opposite, when the number of periodic instances increases, it clearly is the dominating factor when it comes to the performance of Phase-TA, with up to a factor 2 between the execution times of the analysis for 45-minute sub-profiles. Thus, the number of detected periodic instances seems to be the dominating factor regarding the performance of Phase-TA.

From a more down-to-earth perspective, the execution time of Phase-TA seems rather acceptable considering that the analysis of a 45-minute long profile is achieved in about 3 minutes at most (knowing that, given the $5ms$ sampling period, such a profile contains 540000 samples).

Let's now focus on 10-minute long profiles. Indeed, section IV-C showed that, as a rule of thumb, it can be considered that the representative patterns inferred by Phase-TA are relevant for profiles lasting at least 10 minutes. Table III shows that the associated analysis time is lesser than 10 seconds.

Last remark before concluding this section: the performance of Phase-TA regarding the analysis of the profiles associated with the execution of an application **scales linearly with the number of nodes** running the application. Indeed, once again, each profile is specific to a node. Hence, the algorithmic complexity associated with its analysis is **totally independent** of the number of nodes.

As a result, it seems that Phase-TA can be used **on-the-fly** to analyze sub-profiles associated with the **ongoing execution** of an iterative HPC application.

VI. Use cases, motivations, and future work

When it comes to exploiting the representative patterns inferred by Phase-TA, there are three main use cases.

The first one consists in performing dimensionality reduction on time series exhibiting periodicities. Indeed, the inferred patterns could, for instance, be used to sum up a (part of a) profile.

Secondly, the patterns constitute an alternative representation of the time series they are inferred from. And those representations could be more suited than the whole time series for several families of machine learning algorithms, ranging from clustering to time series generation, passing by time series forecasting.

Finally, forecasting the future behavior of an application tends to be a cornerstone of tools resorting to on-the-fly reconfiguration to optimize an objective function. Well, Phase-TA makes it possible to characterize an HPC application being executed, and, from that characterization, to account for its future behavior by leveraging the associated periodicities. Indeed, as it is shown by sections IV-B and V-B, the inferred representative patterns account for a large portion of the execution time of the application, and can be inferred on-the-fly. Hence, it is possible for a reconfiguration tool to build, at runtime, application-specific reconfiguration rules, from the analysis of the inferred representative patterns. These rules can then be applied to a significant portion of the execution time of the application. Additionally, it is important to note that the aforementioned reconfiguration rules are **predictive**, rather than **reactive**, as it is the case for most of the reconfiguration tools based on online analysis. Indeed, thanks to the fact that representative patterns are associated with **periodicities**, the reconfiguration rules inferred from them are built with the knowledge of the whole pattern (including future samples, hence the term *predictive*), not only of the past samples (the reconfiguration is then a *reaction* to past samples).

The latter use case is precisely what motivated us to work on Phase-TA. Indeed, when working on BDPO [2], the need for a representative characterization of the profile of an application arose. In a few words, BDPO resorts to fine-grain monitoring of the IPC profile of an application to optimize the energy efficiency associated with the execution of the latter, while being **completely agnostic** of both the **application**, and the **execution environment** (e.g. programmation paradigm, MPI libraries, ...). To improve energy efficiency, it implements Dynamic Voltage-Frequency Scaling (DVFS) which consists in adapting the Voltage-Frequency functioning point of the cores of the processors to the workload being executed. The idea behind a DVFS controller like BDPO is to downscale the CPU frequency in compute-less (memory and/or IO) phases to minimize energy consumption, and upscale back the CPU frequency in compute-bound phases, while trying to keep performance degradation under control. The experimental methodology presented in [2] is designed to infer an node-specific configuration for BDPO, which enables satisfactory ratios between energy savings and performance degradations, for multiple applications. However, it was empirically verified that even better ratios are achievable with application-specific configurations for BDPO. But crafting those configurations notably requires an expertise the standard HPC user is not likely to have. Hence the need for an analysis tool capable of characterizing the whole profile of an application on-the-fly.

That is why our future work will focus on leveraging the representative patterns inferred by Phase-TA to infer DVFS reconfiguration rules specific to the target application. The first task at hand will consist in designing a rule engine to specify the DVFS opportunities, such as the regions of interest displayed by Fig. 3, so that they may be uniquely detected at the runtime of the executed application. Several paths are to be explored. For instance, extracting and using an excerpt of the representative pattern as a signature of a reconfiguration opportunity, and identify it thanks to DTW measurements between the signature extract and the live profile of the application. Another envisaged solution consists in using generative grammar and constraint programming [14] to define constraints on the live profile of the application, which, when satisfied, trigger a DVFS reconfiguration.

Then, the subject of automating the detection of the DVFS opportunities, given the inferred representative patterns, should be tackled. We envision to develop a tool resorting to the combination of the experimental methodology associated with BDPO [2], and architecture-specific performance models, such as the ones presented by [15], so as to detect node-specific

DFVS opportunities from the analysis of the inferred representative patterns.

VII. Related work

Despite an extensive bibliographic search, the authors could not find existing work dealing specifically with inferring real-valued patterns representing the periodicities of piecewise-periodic time series.

Conversely, a lot of successful research efforts tackled the issue of detecting periodicities in time series, especially in the field of data mining [16]–[20]. For instance, [18] present two convolution-base algorithms to detect both symbol and segment periodicities from time series. To do so, an approach similar to the one implemented by `Phase-TA` is used: it is based on the comparison between the original time series and shifted versions of the latter. However, their methodology requires to discretize the time series, making almost impossible the inference of an accurate real-valued representative pattern.

With a very similar goal, `AUTOPERIOD`, presented in [20], leverages Fourier analysis by combining circular autocorrelation and periodogram to identify the most significative and relevant harmonics of a time series. Used as feature extraction technique, it leads to very accurate clustering of timeseries. Nevertheless, `AUTOPERIOD` does not infer patterns representing the periodicities.

When it comes to leveraging applications' being composed of several types of phases, and DVFS, in order to optimize the energy consumption associated with the execution of an application, several research works can be cited [21]–[24].

Let's lay emphasis on `MREEF` [22], a reconfiguration tool which uses Execution Vectors (EV) to determine which type of phase is associated with the execution of an application at a given time, and based on the said type of phase, enforces appropriate DVFS actions. One EV is generated per second: it consists in a collection of per-cycle rates, over the past second, of several performance counters. Using a set of benchmarks specially designed to exhibit certain types of phases, the authors built a taxonomy of application phases. Then, for each type of phase, they computed a representative EV, called a reference EV. As a consequence, it is possible to determine what kind of application phase is currently executed, by comparing an EV to all the reference EVs. Contrary to `Phase-TA`, neither does `MREEF` detect nor does it characterize the periodicities of the executed application. Furthermore, the reconfigurations it performs are reactive, and rely on the assumption that the next EV should be similar to the currently considered EV (otherwise, the reconfiguration could even be detrimental).

Additionally, it is worth noting that previous research efforts [25]–[28] have demonstrated that significant energy savings can be achieved through approaches based on offline analysis. The workflows of these approaches consist in multiple steps, notably: (1) the profiling of an initial execution of the application, (2) the definition of regions of interest in the source code of the application through annotations, and (3) the analysis and exploitation of the annotations and the generated profile to enforce reconfiguration. While each individual step is performed using automatic tools, the whole process is stil manual and performed by hand. Moreover, the need to annotate the source code to delimit the regions of interest can jeopardize the applicability of such techniques when the access to the source code is impossible.

Finally, let's focus on `EAR` [15]: a richly-featured energy management solution for supercomputers which notably implements `DynAIS`. The latter is an evolution of [4] applied to the detection of symbol periodicity. In a few words, by dynamically overloading the PMPI wrapper through the `LDPRELOAD` environment variable, `EAR` traces the sequences of MPI calls performed by an HPC application. Then, by identifying recurrent patterns of MPI calls, `DynAIS` detects the iterative structure of the application, i.e. its the main loop nest. Thanks to performance models and the monitoring of the identified loop nest through performance counters, `EAR` determines whether or not energy-related optimizations are possible, notably by leveraging DVFS. However, the periodicity detection seems not to be combined with the monitoring information to build representative patterns associated with the identified main loops. Moreover, the approach of `Phase-TA` is agnostic of both the HPC application and the parallel programming environment, whereas `EAR` can only be used with MPI applications (to be precise, it even requires that at least one MPI call belongs to the main loop nest), and overloads the used MPI library, which is not always feasible.

VIII. Conclusion

To sum up this article, it should be highlighted that `Phase-TA` is a black-box analysis tool which detects and characterizes the periodicities associated with the execution of iterative HPC applications. Additionally, it was empirically demonstrated, on a panel of HPC applications, that the detected periodicities are accountable for a very significant part of their execution times.

Furthermore, the representative patterns generated by `Phase-TA` appear to be relevant and accurate proxies for the detected periodicities. On top of that, analysing an extract of an application profile produces relevant representative patterns at the scale of the whole profile, making it possible to infer the aforementioned patterns on-the-fly.

Consequently, the latter can be exploited in several ways, from dimensionality reduction for piecewise-periodic time series, to DVFS opportunities detection.

Our future work will precisely focus on leveraging the patterns inferred by `Phase-TA` to optimize the power-efficiency associated with the execution of iterative HPC applications through predictive, rather than reactive, CPU frequency scaling.

Acknowledgment

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see

https://www.grid5000.fr).
A sincere thank you to Imma Presseguer, and the Atos *Power Efficiency* team, especially Philippe Rols and Julien Forot.

REFERENCES

[1] H. Sakoe and S. Chiba, "A dynamic programming approach to continuous speech recognition," in *Proceedings of the Seventh International Congress on Acoustics, Budapest*, vol. 3, 1971, pp. 65–69.

[2] M. Stoffel and A. Mazouz, "Improving power efficiency through fine-grain performance monitoring in hpc clusters," in *2018 IEEE International Conference on Cluster Computing (CLUSTER) - HPCMASPA Workshop*, 2018, pp. 552–561.

[3] "Home page of `perfmon2`," http://perfmon2.sourceforge.net, last accessed in August 2020.

[4] F. Freitag, J. Corbalan, and J. Labarta, "A dynamic periodicity detector: application to speedup computation," in *Proceedings 15th International Parallel and Distributed Processing Symposium. IPDPS 2001*, 2001.

[5] M. Ankerst, M. M. Breunig, H. peter Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure." ACM Press, 1999, pp. 49–60.

[6] F. Petitjean, A. Ketterlin, and P. Gançarski, "A global averaging method for dynamic time warping, with applications to clustering," *Pattern Recognition*, vol. 44, no. 3, pp. 678–693, 2011.

[7] F. Petitjean, G. Forestier, G. I. Webb, A. E. Nicholson, Y. Chen, and E. Keogh, "Dynamic time warping averaging of time series allows faster and more accurate classification," in *2014 IEEE International Conference on Data Mining*, 2014, pp. 470–479.

[8] "Home page of the Grenoble site of Grid'5000," https://www.grid5000.fr/w/Grenoble:Home, last accessed in August 2020.

[9] "Home page of NEMO," https://www.nemo-ocean.eu/, last accessed in August 2020.

[10] "Home page of HPCG," https://www.hpcg-benchmark.org/, last accessed in August 2020.

[11] "Home page of the OpenFOAM CFD toolbox," https://www.openfoam.com/, last accessed in August 2020.

[12] R. Sibson, "Slink: An optimally efficient algorithm for the single-link cluster method," *The Computer Journal*, vol. 16, no. 1, pp. 30–34, 01 1973.

[13] "Definition of the manhattan distance," https://en.wikipedia.org/wiki/Taxicab_geometry, last accessed in August 2020.

[14] G. Madi Wamba, Y. Li, A.-C. Orgerie, N. Beldiceanu, and J.-M. Menaud, "Cloud workload prediction and generation models," in *SBAC-PAD: International Symposium on Computer Architecture and High Performance Computing*, Campinas, Brazil, Oct. 2017, pp. 89–96.

[15] J. Corbalan and L. Brochard, "Ear: Energy management framework for supercomputers," Available at https://www.bsc.es/sites/default/files/public/bscw2/content/software-app/technical-documentation/ear.pdf, Barcelona Supercomputing Center (BSC), Tech. Rep., 2019, last accessed in April 2020.

[16] E. Keogh, S. Lonardi, and B.-c. Chiu, "Finding surprising patterns in a time series database in linear time and space," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 550–556.

[17] P. Indyk, N. Koudas, and S. Muthukrishnan, "Identifying representative trends in massive time series data sets using sketches," in *VLDB*, 2000, pp. 363–372.

[18] M. G. Elfeky, W. G. Aref, and A. K. Elmagarmid, "Periodicity detection in time series databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 7, pp. 875–887, 2005.

[19] F. Rasheed and R. Alhajj, "STNR: A suffix tree based noise resilient algorithm for periodicity detection in time series databases," *Applied Intelligence*, vol. 32, no. 3, pp. 267–278, 2010.

[20] M. Vlachos, P. Yu, and V. Castelli, "On periodicity detection and structural periodic similarity," in *Proceedings of the 2005 SIAM International Conference on Data Mining*, pp. 449–460.

[21] J. P. Halimi, B. Pradelle, A. Guermouche, and W. Jalby, "Forest-mn: Runtime dvfs beyond communication slack," in *International Green Computing Conference*, Nov 2014, pp. 1–6.

[22] G. L. Tsafack, L. Lefevre, J.-M. Pierson, P. Stolf, and G. Da Costa, "Application-agnostic framework for improving the energy efficiency of multiple hpc subsystems," pp. 62–69, 03 2015.

[23] D. Cesarini, A. Bartolini, P. Bonf, C. Cavazzoni, and L. Benini, "Countdown - three, two, one, low power! a run-time library for energy saving in mpi communication primitives," 06 2018.

[24] C. Isci, G. Contreras, and M. Martonosi, "Live, runtime phase monitoring and prediction on real systems with application to dynamic power management," in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, 2006, pp. 359–370.

[25] M. Kumaraswamy, A. Chowdhury, M. Gerndt, Z. Bendifallah, O. Bouizi, U. Locans, L. Říha, O. Vysockỳ, M. Beseda, and J. Zapletal, "Domain knowledge specification for energy tuning," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 6, 2019.

[26] E. Calore, A. Gabbana, S. F. Schifano, and R. Tripiccione, "Evaluation of DVFS techniques on modern HPC processors and accelerators for energy-aware applications," *Concurr. Comput. Pract. Exp.*, vol. 29, no. 12, 2017.

[27] O. Vysocky, M. Beseda, L. Říha, J. Zapletal, M. Lysaght, and V. Kannan, "Meric and radar generator: Tools for energy evaluation and runtime tuning of hpc applications," in *High Performance Computing in Science and Engineering*. Springer International Publishing, 2018, pp. 144–159.

[28] C. Silvano, G. Agosta, A. Bartolini, A. R. Beccari, L. Benini, L. Besnard, J. Bispo, R. Cmar, J. M. Cardoso, C. Cavazzoni *et al.*, "Supporting the scale-up of high performance application to pre-exascale systems: The antarex approach," in *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 2019, pp. 116–123.